Computer Science 336X
Spring 2012
Homework 6
Due Wednesday, March 7, 11:59 pm

1. Implement the HeightMap class as specified in the header file HeightMap.h. The idea is to generate:

1) an array of vertices for points on the surface defined by a given function \( y = f(x, z) \)
2) a parallel array of normal vectors, one for each vertex, computed by averaging the face normals for the triangles adjacent to the vertex
3) an array of indices that could be used to render the surface using GL_TRIANGLES
4) an array of indices that could be used to render a wireframe using GL_LINES

The idea is to first compute sample values from the function at equally spaced points in the x-z plane. You can imagine these as a 2-d array. In the illustration below, we are looking down the y-axis at the x-z plane, and the parameters numPointsX and numPointsY are both 5 (see the comment preceding the HeightMap constructor).
The red numbers refer to the actual indices of the vertex array, as it should be returned by
GetVertices(), the black numbers are just rows and columns. (Even though this is pictured here
as a 2d array it might be simpler to create it as a 1d array and use a conversion such as index =
row * numPointsX + column to get the index.)

To render as a polygonal mesh using GL_TRIANGLES, the first few indices could be:
0,5,6, 0,6,1, 1,6,7, 1,7,2, etc.

You don’t have to use this exact ordering, just be sure you list the vertices of individual triangles
in counterclockwise order.

To render the wireframe using GL_LINES, the first few indices could be:
0,1, 1,6, 6,5, 5,0 0,6, 1,2, 2,7, 7,6, 6,1 1,7, etc.

To create the normal vectors, start with a parallel array of vec3 the same size as the vertex array
v. Initialize each entry to a zero vector. Iterate through the mesh indices in groups of three.
Each triple is a triangle (a, b, c), where a, b, and c are indices into the array. Use a cross product
to create an outward-directed normal vector for the triangle, e.g. (v[b] – v[a]) X (v[c] – v[a]).
Add that normal to the same three array locations a, b, and c in the normal array. After all
triangles are generated, go through and normalize all the vectors in the normal array. Each one
will be the average of the normal vectors for all triangles that are adjacent to that vertex.

The header file includes a function GetTexCoords(). For this homework you can just return null.

**Trying it out**

There is some sample code in the solution light.sln. (This is a complete VS solution, not just a
project, since the header files for RenderUtils are added to the include directory and
RenderUtils.lib is added to the lib directory.) There is some sample code provided by Jonathan
in a project called height_map_example. *This project will not build until you create a skeleton
for HeightMap.cpp and add it to the project*. It includes a main.cpp and a copy of HeightMap.h.
Note that it uses the Camera, Shader, and VertexArray classes that we discussed in class. It is
already set up to statically link RenderUtils.lib.

Edit the init() function to choose a function to try. Initially it displays a wireframe using a shader
that just determines a vertex color from the y-value. Vertices with the smallest y-value are black,
and those with the largest y-value are white. The minimum and maximum y-values are
determined by your HeightMap and are set as uniforms in the shader.)
Use the ‘p’ key to toggle between phong shading and ‘height’ shading. Use the ‘m’ key to toggle between wireframe and polygonal mesh rendering. You’ll need to have the normal vectors implemented for the latter to work.